Applied Mathematics Letters 23 (2010) 272–276

Contents lists available at ScienceDirect

Applied Mathematics Letters

journal homepage: www.elsevier.com/locate/aml

Applications of propositional logic to workflow analysis☆

Glória Cravo

Departamento de Matemática e Engenharias, Universidade da Madeira, 9000-390 Funchal, Madeira, Portugal

ARTICLE INFO

Article history:
Received 3 April 2009
Received in revised form 14 October 2009
Accepted 19 October 2009

Keywords:
Graphs
Classical propositional logic
Workflows
Process modeling
Business processes

ABSTRACT

In this paper our main goal is to describe the structure of workflows. A workflow is an abstraction of a business process that consists of one or more tasks to be executed to reach a final objective. In our approach we describe a workflow as a graph whose vertices represent workflow tasks and the arcs represent workflow transitions. Moreover, every arc (t_k, t_l) (i.e., a transition) has attributed a Boolean value to specify the execution/non-execution of tasks t_k, t_l. With this attribution we are able to identify the natural flow in the workflow.

Finally, we establish a necessary and sufficient condition for the termination of workflows. In other words, we identify conditions under which a business process will be complete.

1. Introduction

In this paper, we use graph theory and propositional logic to describe and analyze workflows. In particular, the use of propositional logic is a fundamental instrument to determine if a workflow has been correctly designed by an end user from the termination point of view. A workflow is an abstraction of a business process that consists of one or more tasks that need to be executed to complete a process (for example, hiring process, sales order processing, article reviewing, member registration, etc.), that can include human activity and/or software applications to carry out activities. A workflow can be represented by a graph, whose tasks are represented with vertices and the tasks are modeled with arcs, known as transitions. Each task represents a unit of work to be executed either by humans or application programs. A workflow describes all of the tasks needed to achieve each step in a business process.

Workflows may involve many distinct, heterogeneous, autonomous, and distributed tasks that are interrelated in complex ways. The complexity of large workflows requires a precise modeling to ensure that they perform according to initial specifications.

A vast number of formal frameworks have been proposed to allow workflow modeling verification and analysis, such as State and Activity Charts [1], Graphs [2], Event-Condition-Action rules [3,4], Petri Nets [5–8], Temporal Logic [9] and Markov chains [10]. Other approaches can be found in [11–13].

In this paper our formalism is based on graph theory and propositional logic. One relevant aspect of our approach is the use of propositional logic. In particular, the attribution of Boolean values to each arc of the workflow is very important, since it allows us to identify the natural flow in the workflow.

Finally, we identify conditions under which a workflow logically terminates. In other words, we are able to verify if a business process will be complete.

2. Workflow analysis

In this section we analyze the structure of workflows. We start by presenting the formal concept of a workflow.

☆ This research was done within the activities of the *Centro de Estruturas Lineares e Combinatórias.*
E-mail address: gcravo@uma.pt.

0893-9659/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.aml.2009.10.007

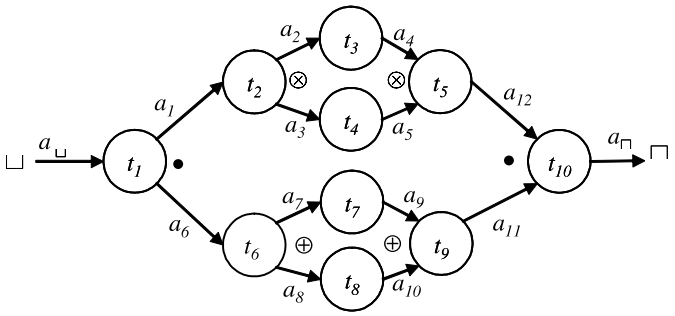

Fig. 1. Example of a workflow.

Definition 1. A *workflow* is a tri-logic acyclic directed graph $WG = (T, A)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite nonempty set of vertices representing workflow tasks. Each task t_i (i.e., a vertex) has an input logic operator (represented by $\succ t_i$) and an output logic operator (represented by $t_i \prec$). An input/output logic operator can be the logical AND ($\bullet$), the OR ($\otimes$), or the XOR-exclusive-or-($\oplus$). The set $A = \{a_{\sqcup}, a_{\sqcap}, a_1, a_2, \ldots, a_m\}$ is a finite nonempty set of arcs representing workflow transitions. Each transition $a_i, i \in \{1, \ldots, m\}$, is a tuple (t_k, t_l) where $t_k, t_l \in T$. The transition $a_{\sqcup}$ is a tuple of the form $(\sqcup, t_1)$ and transition $a_{\sqcap}$ is a tuple of the form $(t_n, \sqcap)$. The symbols $\sqcup$ and $\sqcap$ represent abstract tasks which indicate the entry and ending point of the workflow, respectively. We use the symbol $'$ to reference the label of a transition, i.e., a_i' references transition a_i, $a_i \in A$. The elements a_i' are called Boolean terms and form the set A'.

Example 2. In Fig. 1 is shown a workflow $WG = (T, A)$, where $T = \{t_1, t_2, \ldots, t_{10}\}, A = \{a_{\sqcup}, a_{\sqcap}, a_1, a_2, \ldots, a_{12}\}$ and $A' = \{a_{\sqcup}', a_{\sqcap}', a_1', a_2', \ldots, a_{12}'\}$. The tuple $a_2 = (t_2, t_3)$ is an example of a transition. In task t_{10}, the input logic operator ($\succ t_{10}$) is an AND ($\bullet$); in task t_2 the output logic operator ($t_2 \prec$) is an *OR* ($\otimes$).

Definition 3. For any task $t_i \in T$, the *incoming transitions* are the tuples of the form $a_j = (x, t_i), x \in T, a_j \in A$, and the *outgoing transitions* are the tuples of the form $a_l = (t_i, y), y \in T, a_l \in A$.

Example 4. In Fig. 1, the incoming transition for task t_2 is $a_1 = (t_1, t_2)$ and the outgoing transitions are $a_2 = (t_2, t_3)$ and $a_3 = (t_2, t_4)$.

Definition 5. Given any task $t_i \in T$, the *incoming condition* is the Boolean expression $a_{k_1}' \varphi \ldots \varphi a_{k_l}', \varphi \in \{\bullet, \otimes, \oplus\}$, where the terms $a_{k_1}', \ldots, a_{k_l}' \in A'$, and $a_{k_1}, \ldots, a_{k_l}$ are the incoming transitions of task t_i. The terms $a_{k_1}', \ldots, a_{k_l}'$ are connected with the logical operator $\succ t_i$. If the task has only one incoming transition then the condition does not have logical operator.

The *outgoing condition* for task t_i is the Boolean expression $a_{k_1}' \varphi \ldots \varphi a_{k_l}', \varphi \in \{\bullet, \otimes, \oplus\}$, where the terms $a_{k_1}', \ldots, a_{k_l}' \in A'$, and $a_{k_1}, \ldots, a_{k_l}$ are the outgoing transitions of task t_i. The terms $a_{k_1}', \ldots, a_{k_l}'$ are connected with the logical operator $t_i \prec$. If the task has only one outgoing transition then the condition does not have logical operator.

Example 6. Consider task t_2 in Fig. 1. Its incoming condition is a_1' and its outgoing condition is $a_2' \otimes a_3'$.

A workflow is a set of tasks and transitions. The tasks can be considered as atomic pieces of the workflow, since they generate all transitions. Clearly, knowing the tasks and transitions of the workflow, allows to know the precise structure of the workflow. However, we need to exploit under which conditions an arbitrary task is executed and the consequences of its execution, i.e., we need to determine the natural flow of the workflow. Notice that when a workflow is correctly designed, it terminates by enabling the ending transition $a_{\sqcap}$. Our main goal, is to identify conditions under which the ending transition $a_{\sqcap}$ is enabled, i.e., the workflow is correctly designed.

In order to analyze the consequences of the execution of a certain task, we introduce the concept of Event–Action model.

Definition 7. Let $WG = (T, A)$ be a workflow and let $t_i \in T$. *An Event–Action (EA) model* for task t_i is an implication of the form $t_i : f_E \rightsquigarrow f_C$, where f_E and f_C are the incoming and outgoing conditions of task t_i, respectively. An *EA* model has the behavior with two distinct modes: when f_E is evaluated to *true*, f_C is also evaluated to *true*; when f_E is evaluated to *false*, f_C is always *false*. The condition f_E is called the *event condition* and f_C is called the *action condition*.

Every *EA* model $t_i : f_E \rightsquigarrow f_C$ has attributed a Boolean value, according to the following rules:

(i) If both f_E, f_C are *true*, then its Boolean value is *true*;
(ii) If both f_E, f_C are *false*, then its Boolean value is *false*.

A workflow starts its execution when transition $a_{\sqcup}$ is enabled. A transition is enabled/disabled if the respective Boolean term is asserted to be true/false. Thus, the workflow starts its execution by asserting $a_{\sqcup}'$ to be true.

For any *EA* model, the incoming condition propagates its Boolean value to the respective outgoing condition, i.e., the Boolean value of the outgoing condition is not arbitrary, it always depends on the Boolean value of the incoming condition, according to Definition 7. In other words, an *EA* model has a behavior with two distinct modes: when f_E is evaluated to *true*

and when f_E is evaluated to *false*. In the first situation, its outgoing transitions are enabled or disabled in such a way that f_C is *true*; in the other case, all the outgoing transitions are disabled and, consequently, f_C becomes *false*.

The behavior of the *EA* models is very important, since the workflow is a set of *EA* models and the complete execution of the workflow depends on the execution of all its *EA* models. Besides, every *EA* model propagates its behavior to the following *EA* models connected to it. We need to exploit how this propagation affects the execution of the workflow. Therefore, we will focus our study on the *EA* models.

Definition 8. Given $t_i : f_E \rightsquigarrow f_C$ an arbitrary *EA* model of *WG*, we say that the model is *positive* if its Boolean value is *true*, otherwise the model is said to be *negative*.

A workflow starts its execution by enabling transition $a_{\sqcup}$. Any transition can be enabled explicitly by an user or implicitly by an external event. Note that the outgoing conditions are enabled only after the respective incoming conditions being enabled.

Since the behavior of a workflow is determined by its *EA* models, a natural concern, is the exhaustive study of the *EA* models. Next we define two different types of *EA* models.

Definition 9. An *EA* model $t_u : f_E \rightsquigarrow f_C$ is said to be *simple* if $f_E = a'_i$ and $f_C = a'_j$, $i, j \in \{\sqcup, \sqcap, 1, \ldots, m\}$, with $i \neq j$. Otherwise, the *EA* model is said to be *compound*.

Example 10. The *EA* model $t_3 : a'_2 \rightsquigarrow a'_4$, from Fig. 1, is simple. The *EA* models $t_2 : a'_1 \rightsquigarrow a'_2 \otimes a'_3$ and $t_9 : a'_9 \oplus a'_{10} \rightsquigarrow a'_{11}$, from Fig. 1, are compound.

The study of simple *EA* models is very easy: once the incoming transition is enabled, necessarily its outgoing transition is enabled. The analysis of compound *EA* models is more difficult. In this case, the Boolean value of the event/action condition will depend on the Boolean terms associated with its incoming/outgoing transitions, which can be *true* or *false*. Obviously, the Boolean value of the outgoing condition also depends on the Boolean value of the incoming condition. The analysis of the compound *EA* models is fundamental. Notice the complete execution of the workflow will depend on the execution of these *EA* models.

Now we will focus our study on the logical termination of workflows. The logical termination is an important structural property that allows to check if a workflow finishes by enabling transition $a_{\sqcap}$. For example, it allows to know previously if a business process, such as a loan application, will be complete.

Once $a_{\sqcup}$ is enabled, tasks of the workflow start their execution. The processing of the workflow stops when one of the following cases occurs:

(a) The workflow finishes by enabling transition $a_{\sqcap}$;
(b) The processing stops at some task t_i, $i \in \{2, \ldots, n-1\}$.

If a workflow is correctly designed, necessarily case (a) is satisfied. Our aim is to identify conditions under which this case occurs. For that purpose we introduce the formal concept of logical termination.

Definition 11. Let $WG = (T, A)$ be a workflow. We say that *WG logically terminates* if $a'_{\sqcap}$ is *true* whenever $a'_{\sqcup}$ is *true*.

Proposition 12. *Let $WG = (T, A)$ be a workflow. If all EA models of WG are simple, then WG always logically terminates.*

Proof. Clearly, if all *EA* models of the workflow are simple, then its structure is the following:

$$\sqcup \xrightarrow{a_{\sqcup}} t_1 \xrightarrow{a_1} t_2 \xrightarrow{a_2} t_3 \ldots t_{n-1} \xrightarrow{a_{n-1}} t_n \xrightarrow{a_{\sqcap}} \sqcap.$$

In this case, the set of compound *EA* models is empty. Assuming that $a'_{\sqcup}$ is *true*, necessarily a'_1 is *true*. This fact implies that a'_2 is *true*. Recursively, for any transition of *WG*, the respective Boolean term is *true*. In particular, we can infer that $a'_{\sqcap}$ is *true*. Hence $a'_{\sqcap}$ is *true*, whenever $a'_{\sqcup}$ is *true*, which means that *WG* logically terminates. ■

Our concern is to study the general situation, when the set of compound *EA* models of *WG* is nonempty. For that purpose, we start by introducing the concept of materialized workflow instance.

Definition 13. A *materialized workflow instance* of *WG* is an assignment of Boolean values to all Boolean terms $a'_j \in A'$, according to Definition 7.

Our main result is the following, where we provide a necessary and sufficient condition for the logical termination of workflows.

Theorem 14. *Let $WG = (T, A)$ be a workflow. Then WG logically terminates if and only if, for any materialized workflow instance, all its compound EA models are positive.*

Proof. Let $WG = (T, A)$ be a workflow and assume that *WG* logically terminates. Suppose by contradiction, that there exists at least a materialized workflow instance for which there exists a negative compound *EA* model. According to Definition 7,

since $a'_{\sqcup}$ is asserted to be *true*, then f_{C_1} is also *true*. So, there exists $j \in \{2, \ldots, n\}$ such that the compound *EA* model $t_j : f_{E_j} \rightsquigarrow f_{C_j}$ is negative. Hence, both conditions f_{E_j}, f_{C_j}, are *false*. Therefore, for every $k > j$ if $t_k : f_{E_k} \rightsquigarrow f_{C_k}$ is a compound *EA* model connected to the *EA* model $t_j : f_{E_j} \rightsquigarrow f_{C_j}$, necessarily $t_k : f_{E_k} \rightsquigarrow f_{C_k}$ is also negative. As a consequence, it follows that $a_{\sqcap}$ is not enabled, i.e., $a'_{\sqcap}$ is *false*, which is a contradiction, since the workflow logically terminates. Thus, for every materialized workflow instance, all compound *EA* models of *WG* are positive.

Conversely, suppose that the workflow starts its execution, i.e., $a'_{\sqcup}$ is asserted to be *true*.

Clearly if $t_n : f_{E_n} \rightsquigarrow a'_{\sqcap}$ is a compound *EA* model, according to the hypothesis this *EA* model is positive and consequently, $a'_{\sqcap}$ is *true*.

From now on, we may assume that $t_n : f_{E_n} \rightsquigarrow a'_{\sqcap}$ is a simple *EA* model. Now, we consider the *EA* model $t_{n-1} : f_{E_{n-1}} \rightsquigarrow a'_m$. If $t_{n-1} : f_{E_{n-1}} \rightsquigarrow a'_m$ is a compound *EA* model, then according to the hypothesis this *EA* model is positive. Therefore, both $f_{E_{n-1}}, a'_m$ are *true*. Since $f_{E_n} = a'_m$, and a'_m is *true*, then according to Definition 7, $a'_{\sqcap}$ is also *true*.

If $t_{n-1} : f_{E_{n-1}} \rightsquigarrow a'_m$ is a simple *EA* model, using similar arguments we can infer the existence of a positive integer $l \in \{2, \ldots, n-2\}$ such that $t_l : f_{E_l} \rightsquigarrow f_{C_l}$ is compound and for every $j > l$, $t_j : f_{E_j} \rightsquigarrow f_{C_j}$ is simple, if t_j is connected to t_l. Notice that according to the hypothesis, the *EA* model $t_l : f_{E_l} \rightsquigarrow f_{C_l}$ is positive. Therefore, both conditions f_{E_l}, f_{C_l} are *true*. Now, according to Definition 7, it follows that for every $j > l$, if $t_j : f_{E_j} \rightsquigarrow f_{C_j}$ is connected to $t_l : f_{E_l} \rightsquigarrow f_{C_l}$, necessarily $t_j : f_{E_j} \rightsquigarrow f_{C_j}$ is also positive, which means that both incoming and outgoing conditions are *true*. Consequently, $a'_{\sqcap}$ is *true*.

Since $a'_{\sqcap}$ is asserted to be *true*, we can conclude that *WG* logically terminates. ■

Example 15. Let us consider the workflow from Fig. 1 and assume that $a'_{\sqcup}$ is true. The compound *EA* models of *WG* are the following: $a'_{\sqcup} \rightsquigarrow a'_1 \bullet a'_6$, $a'_1 \rightsquigarrow a'_2 \otimes a'_3$, $a'_4 \otimes a'_5 \rightsquigarrow a'_{12}$, $a'_6 \rightsquigarrow a'_7 \oplus a'_8$, $a'_9 \oplus a'_{10} \rightsquigarrow a'_{11}$, $a'_{11} \bullet a'_{12} \rightsquigarrow a'_{\sqcap}$.

In this situation, the materialized workflow instances of *WG* are the following:

(1) $a'_{\sqcup} = a'_1 = a'_6 = \mathit{true}$, $a'_2 = \mathit{true}$, $a'_3 = \mathit{true}$, $a'_2 = a'_4 = \mathit{true}$, $a'_3 = a'_5 = \mathit{true}$, $a'_{12} = \mathit{true}$, $a'_7 = \mathit{true}$, $a'_8 = \mathit{false}$, $a'_7 = a'_9 = \mathit{true}$, $a'_8 = a'_{10} = \mathit{false}$, $a'_{11} = \mathit{true}$, $a'_{\sqcap} = a'_{11} = a'_{12} = \mathit{true}$.

(2) $a'_{\sqcup} = a'_1 = a'_6 = \mathit{true}$, $a'_2 = \mathit{true}$, $a'_3 = \mathit{true}$, $a'_2 = a'_4 = \mathit{true}$, $a'_3 = a'_5 = \mathit{true}$, $a'_{12} = \mathit{true}$, $a'_7 = \mathit{false}$, $a'_8 = \mathit{true}$, $a'_7 = a'_9 = \mathit{false}$, $a'_8 = a'_{10} = \mathit{true}$, $a'_{11} = \mathit{true}$, $a'_{\sqcap} = a'_{11} = a'_{12} = \mathit{true}$.

(3) $a'_{\sqcup} = a'_1 = a'_6 = \mathit{true}$, $a'_2 = \mathit{true}$, $a'_3 = \mathit{false}$, $a'_2 = a'_4 = \mathit{true}$, $a'_3 = a'_5 = \mathit{false}$, $a'_{12} = \mathit{true}$, $a'_7 = \mathit{true}$, $a'_8 = \mathit{false}$, $a'_7 = a'_9 = \mathit{true}$, $a'_8 = a'_{10} = \mathit{false}$, $a'_{11} = \mathit{true}$, $a'_{\sqcap} = a'_{11} = a'_{12} = \mathit{true}$.

(4) $a'_{\sqcup} = a'_1 = a'_6 = \mathit{true}$, $a'_2 = \mathit{true}$, $a'_3 = \mathit{false}$, $a'_2 = a'_4 = \mathit{true}$, $a'_3 = a'_5 = \mathit{false}$, $a'_{12} = \mathit{true}$, $a'_7 = \mathit{false}$, $a'_8 = \mathit{true}$, $a'_7 = a'_9 = \mathit{false}$, $a'_8 = a'_{10} = \mathit{true}$, $a'_{11} = \mathit{true}$, $a'_{\sqcap} = a'_{11} = a'_{12} = \mathit{true}$.

(5) $a'_{\sqcup} = a'_1 = a'_6 = \mathit{true}$, $a'_2 = \mathit{false}$, $a'_3 = \mathit{true}$, $a'_2 = a'_4 = \mathit{false}$, $a'_3 = a'_5 = \mathit{true}$, $a'_{12} = \mathit{true}$, $a'_7 = \mathit{true}$, $a'_8 = \mathit{false}$, $a'_7 = a'_9 = \mathit{true}$, $a'_8 = a'_{10} = \mathit{false}$, $a'_{11} = \mathit{true}$, $a'_{\sqcap} = a'_{11} = a'_{12} = \mathit{true}$.

(6) $a'_{\sqcup} = a'_1 = a'_6 = \mathit{true}$, $a'_2 = \mathit{false}$, $a'_3 = \mathit{true}$, $a'_2 = a'_4 = \mathit{false}$, $a'_3 = a'_5 = \mathit{true}$, $a'_{12} = \mathit{true}$, $a'_7 = \mathit{false}$, $a'_8 = \mathit{true}$, $a'_7 = a'_9 = \mathit{false}$, $a'_8 = a'_{10} = \mathit{true}$, $a'_{11} = \mathit{true}$, $a'_{\sqcap} = a'_{11} = a'_{12} = \mathit{true}$.

Conclusion: for any materialized workflow instance, all compound *EA* models of *WG* are positive. Therefore, the workflow logically terminates.

Notice that the previous example corresponds to the following real situation. Indeed, the workflow from Fig. 1 can represent the tasks needed to be executed to an author take the decision of participating in a conference. Let us assume that tasks t_i, $i \in \{1, \ldots, n\}$ have the following meanings:

t_1	Presentation of a contributed talk;
t_2	Travel reservation;
t_3	Web reservation;
t_4	Reservation by telephone;
t_5	Confirmation/non-confirmation of the reservation;
t_6	Submission of an abstract;
t_7	Acceptance of the abstract;
t_8	Rejection of the abstract;
t_9	Notification of the acceptance/rejection of the abstract;
t_{10}	Author's decision regarding the participation in the conference.

Clearly, the presentation of a contributed talk in a conference implies a travel reservation and the submission of an abstract. The travel reservation can be made by web and/or by telephone. The execution of at least one of these tasks implies the notification of the confirmation/non-confirmation of the reservation.

On the other hand, the submission of an abstract implies necessarily the execution of only one of the tasks: acceptance of the abstract or its rejection. Clearly, the execution of either one of these tasks implies the notification concerning the acceptance/rejection.

Hence, the decision of the author regarding the participation in the conference depends on the execution of both tasks: Confirmation/non-confirmation of the reservation and Notification of the acceptance/rejection of the abstract. It is clear that the author can only decide to participate or not in the conference after those two tasks being executed.

Many other examples can be given. Notice this subject is well known on our everyday experience. Indeed, too many situations in our life can be described by workflows. For example, the request for a credit card, or a loan application are simple examples of workflows.

3. Concluding remarks

In this paper we provide a theoretical mathematical foundation, based on graph theory and propositional logic, that can describe the structure of workflows. One relevant aspect of our approach is the use of propositional logic. In particular the attribution of Boolean values to each transition is one highlight of our study. This attribution allows to identify the natural flow in the workflow.

Finally, our approach allows us to determine under which conditions a workflow will be completed, i.e., a workflow logically terminates. Indeed, we prove that a workflow logically terminates, if and only if, for any materialized workflow instance, all its compound *EA* models are positive.

References

[1] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, K. Dittrich, Enterprise-wide workflow management based on state and activity charts, in: A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (Eds.), Proceedings NATO Advanced Study Institute on Workflow Management Systems and Interoperability, Springer-Verlag, 1998.
[2] METEOR, (Managing End-To-End OpeRations) Project home page, LSDIS Lab, 2006. Acessed from http://lsdis.cs.uga.edu/projects/past/METEOR/.
[3] U. Dayal, M. Hsu, R. Ladin, Organizing long-running activities with triggers and transactions, in: ACM SIGMOD International Conference on Management of Data Table of Contents, Atlantic City, NJ, ACM Press, NY, USA, 1990, pp. 204–214.
[4] J. Eder, H. Groiss, H. Nekvasil, A workflow system based on active databases, in: G. Chroust, A. Benczur (Eds.), Proceedings of CON' 94, Workflow Management: Challenges, Paradigms and Products, Linz, Austria, 1994, pp. 249–265.
[5] W.V.M.P.v.d. Aalst, The application of petri nets to workflow management, Journal of Circuits, Systems and Computers 8 (1) (1998) 21–66.
[6] W.V.M.P.v.d. Aalst, Workflow verification: Finding control-flow errors using petri-net-based techniques, in: W.V.M.P.v.d. Aalst, J. Desel, A. Oberweis (Eds.), Business Process Management: Models, Techniques, and Empirical Studies, Springer-Verlag, Berlin, 2000, pp. 161–183.
[7] F. Gottschalk, W.V.M.P.v.d. Aalst, M.H. Jasen-Vullers, H.M.W. Verbeek, Protor2CPN: Using colored petri nets for configuring and testing business processes, International Journal on Software Tools for Technology Transfer 10 (1) (2008) 95–111.
[8] A. Rozinat, R.S. Mans, M. Song, W.V.M.P.v.d. Aalst, Discovering colored petri nets from event logs, International Journal on Software Tools for Technology Transfer 10 (1) (2008) 57–74.
[9] P. Attie, M. Singh, A. Sheth, M. Rusinkiewicz, Specifying and enforcing intertask dependencies, in: Proceedings of 19th International Conference on Very Large Data Bases, Morgan Kaufman, Dublin, Ireland, 1993, pp. 134–145.
[10] J. Klingemann, J. Wäsch, K. Aberer, Deriving service models in cross-organizational workflows. in: Proceedings of RIDE-Information Technology for Virtual Enterprises, RIDE-VE' 99, Sydney, Australia, 1999, pp. 100–107.
[11] J. Cao, C. Chan, K. Chan, Workflow analysis for web publishing using a state-activity process model, Journal of Systems and Software 76 (3) (2005) 221–235.
[12] F. Gottschalk, W.V.M.P.v.d. Aalst, M.H. Jansen-Vullers, M. La Rosa, Configurable workflow models, International Journal of Cooperative Information Systems 17 (2) (2008) 223–255.
[13] H.M.W. Verbeek, W.V.M.P.v.d. Aalst, A.H.M. Hofstede, Verifying workflows with cancellation regions and OR-joins: An approach based on relaxed soundness and invariants, Computer Journal 50 (3) (2007) 294–314.